Redakcja Elektroniki Praktycznej dziękuje firmie NDN z Warszawy, autoryzowanemu dystrybutorowi i serwisowi firmy Rigol, za udostępnienie oscyloskopu Rigol DS2202 dla potrzeb tego artykułu.

Analiza protokołów (1)

Termin „analiza protokołów" znany jest elektronikom nie od dziś. Specjaliści różnych dziedzin interpretują go jednak zgoła odmiennie. Inżynier telekomunikacji zajmujący się systemami łączności radiowej czy telefonii komórkowej analizę protokołów będzie rozumiał inaczej niż informatyk, a jeszcze inaczej konstruktor projektujący układy elektroniczne. W artykule zajmiemy się protokołami wykorzystywanymi w popularnych interfejsach komunikacyjnych.

Do redakcji naszego pisma przychodzi wiele pytań dotyczących analizy protokołów. Czytelnicy coraz częściej spotykają się z tym terminem w artykułach drukowanych i umieszczanych w Internecie. Temat ten jest jednak poruszany tylko przy okazji recenzji oscyloskopów i analizatorów stanów logicznych. Postanowiliśmy więc pomóc w rozwikłaniu ewentualnych wątpliwości publikując krótki cykl artykułów na ten temat.

Definicja

Już z informacji podanych we wstępie do artykułu wynika, że termin „analiza protokołów" ma wiele znaczeń. Nas będą interesowały protokoły komunikacyjne stosowane na najniższym poziomie systemów elektronicznych, a więc wykorzystywane do łączności między poszczególnymi blokami funkcjonalnymi urządzenia lub bezpośrednio współpracujących ze sobą urządzeń. Bloki te muszą oczywiście charakteryzować się odpowiednią inteligencją, którą zapewniają mikroprocesory, mikrokontrolery, układy programowalne itp.

Szereg protokołów, o których mowa opracowano z uwzględnieniem charakterystycznych cech urządzeń, dla których są dedykowane, inne zoptymalizowano ze względu na media wykorzystywane do łączności lub charakterystyczną zawartość przesyłanych informacji (danych).

Liczba obecnie stosowanych protokołów jest dość duża, ale trzeba pamiętać, że większość z nich powstała w ostatnich 30 latach, a nawet później. Przykładem niech będzie interfejs CAN, którego początki sięgają lat osiemdziesiątych XX wieku. Interfejs ten powstał w wyniku zapotrzebowania na rozwiązania elektroniczne w nowoczesnych samochodach tamtego okresu, a było to wówczas możliwe ze względu na dostateczny rozwój technologii (przede wszystkim podzespołów elektronicznych takich jak mikroprocesory, miniaturowe czujniki wielkości nieelektrycznych itp.). Innym przykładem może być interfejs (i protokół) ARINC 429 wykorzystywany dla odmiany w lotnictwie. W obu przypadkach stosowane są inne rozwiązania układowe, inaczej są zbudowane interfejsy i w odmienny sposób zorganizowano transmisję danych, czyli przyjęto różne protokoły. Jest tak, mimo zbliżonych funkcji realizowanych przez oba interfejsy. Podstawowym zadaniem jest przecież zbieranie danych z licznych czujników rozmieszczonych w wielu punktach samochodu lub samolotu i przesyłanie ich do komputera pokładowego. Ten z kolei na podstawie odebranych informacji wypracowuje sygnały dla elementów wykonawczych (poduszki powietrzne, systemy wspomagania, systemy kontroli trakcji w samochodach czy wyświetlacz komputera pokładowego w samolocie). Można wymienić jeszcze szereg podobnych przykładów, ale w dalszej części artykułu skupimy się na czterech najczęściej stosowanych protokołach stosowanych w interfejsach komunikacyjnych o tej samej nazwie. Będą to:

- **RS-232** – historycznie jeden z pierwszych protokołów komunikacyjnych. Był on wykorzystywany przez szereg lat niemal we wszystkich urządzeniach elektronicznych, niezależnie od ich funkcji i zastosowań (urządzenia przemysłowe, sprzęt wojskowy, medyczny itp.). Jego popularność gwałtownie spadła po opublikowaniu specyfikacji interfejsu USB. Dzisiaj USB zastąpił niemal całkowicie wysłużonego RS-232, ale ze względu na dużą liczbę sprawdzonych i nadal działających aplikacji, RS-232 jest nadal instalowany w wielu urządzeniach.
- **I²C** – protokół opracowany przede wszystkim do komunikacji między mikrokontrolerem a rozmaitymi czujnikami, np. wielkości nieelektrycznych, pamięciami EEPROM itp. Częściowo rozwiązywał on problem ograniczonej szybkości transmisji w RS-232. Inną zaletą protokołu I²C jest wbudowany mechanizm adresowania urządzeń, co znacznie ułatwia selektywną komunikację z wieloma nadajnikami i odbiornikami. Ze względu na prawa licencyjne, spotykane są inne nazwy interfejsu I²C (np. TWI, IIC), jednak w każdym przypadku stosowany jest identyczny protokół.
- **SPI –** obecnie jeden z częściej stosowanych interfejsów komunikacyjnych. Pełni on funkcje zbliżone do omawianych wcześniej RS-232, i I²C. W interfejsie SPI, podobnie jak w I²C, zastosowano transmisję synchroniczną, co wymagało dołożenia dodatkowej linii dla przebiegu taktującego przesyłanie danych.
- **Parallel** – w tym przypadku protokół należy utożsamiać w zasadzie bezpośrednio z samym interfejsem. Dane są przesyłane równolegle przez wieloprzewodowe szyny danych/adresowych. Mimo oczekiwanych dużych szybkości transferu – w jednym takcie przesyłane są całe słowa, interfejs ten nie jest jednak stosowany zbyt często, głównie ze względu na znaczną komplikację połączeń i generowanie sporych zakłóceń.

Pod pojęciem „analiza protokołu” będziemy rozumieli badanie transmisji realizowanej przez jeden z wymienionych interfejsów. Wynikiem takiego pomiaru będzie graficzna prezentacja wysyłanych danych (w postaci oscylogramów) uzupełniona o wyświetlenie zdekodowanych danych. W zależności od charakteru przesłanych informacji można wybrać formę prezentacji danych, np. ASCII, liczby binarne, dziesiętne lub szesnastkowe. Obok tych informacji mogą być ponadto wyświetlane informacje dodatkowe, np. znacznik początku i końca pakietu, znaczniki błędów, adresy urządzeń itp.

Narzędzia diagnostyczne

W naszym kursie ograniczyliśmy się do protokołów wymienionych wcześniej. Przyjęliśmy, że wszystkie pomiary będą wykonywane oscyloskopem Rigol DS2202 z zainstalowaną opcją analizy protokołów. Zastosowana w nim funkcja pomiarowa „Decode” uwzględnia właśnie wymienione protokoły. Niestety, w zestawie brakuje np. USB czy 1-wire. Należy ponadto liczyć się z różnymi rozwiązaniami, które można spotkać u poszczególnych producentów. W oscyloskopach wyższej klasy mamy prawo spodziewać się szerszej gamy obsługiwanych protokołów. Warto również pamiętać, że funkcja analizy protokołów jest też na ogół dostępna w analizatorach stanów logicznych. W każdym przypadku należy upewniać się, czy jest ona instalowana domyślnie czy tylko opcjonalnie za dodatkową opłatą. Pojawiają się już wprawdzie pierwsze przypadki standardowego implementowania analizy przynajmniej najbardziej popularnych protokołów, ale proces ten przebiega z dziwnie dużymi oporami. Miejmy nadzieję, że wkrótce, kupując niemal każdy oscyloskop cyfrowy, od razu będziemy mieli możliwość analizy przynajmniej kilku, np. RS-232 (UART), SPI, I²C, Parallel. Podobną sytuację mieliśmy z analizą widmową FFT. Kiedyś był to rarytas, dzisiaj funkcję tę spotykamy w każdym oscyloskopie cyfrowym.

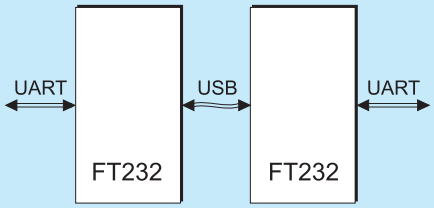
Wirtualne wcielenie analizatora protokołów

Jak mogliśmy się już przekonać, urządzenie o nazwie analizator protokołów w zasadzie nie istnieje. Używając tego terminu zawsze więc będziemy mieli na myśli określoną funkcjonalność na przykład oscyloskopu cyfrowego.

Protokół RS-232 (UART)

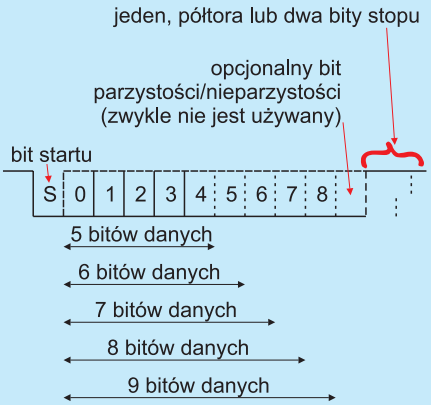
Po niezbędnym wstępie przystępujemy do zajęć praktycznych. Zaczynamy od protokołu RS-232, najczęściej chodzi o jego specyfikację RS-232C. W dostępnych materiałach Czytelnicy mogą spotkać się również z nazwą UART. W obu przypadkach chodzi o ten sam protokół (w sensie ramki transmisyjnej), jednak w uarcie rozpatrywanym jako interfejs mamy do czynienia z inną logiką sygnałów. Poziomy logiczne interfejsu UART odpowiadają technologii zastosowanych układów cyfrowych: TTL, TTL-LS, CMOS, TTL-LVC (3,3 V) itp. Na przykład w technologii TTL-LVC „1” logicznej odpowiada napięcie 3,3 V, a „0” odpowiada 0 V. Jest to bardzo istotne, gdyż w specyfikacji RS-232 przyjęto, że stanowi „1” odpowiada napięcie z zakresu od –3...–15 V, a stanowi „0” napięcie 3...15 V.

Mimo popularności wspomnianego wcześniej interfejsu USB, elektronicy chętnie omijają go stosując popularne mostki USB-UART, np. produkcji FTDI. Pozwala to wykorzystywać procedury transmisyjne identyczne jak dla interfejsu RS-232 (UART). Badając urządzenia z takimi rozwiązaniami zwykle nie ma potrzeby wnikania w transmisję od trony USB, wystarczy analiza protokołu po stronie UART (**rysunek 1**). Oscyloskop Rigol DS2202 jest wówczas wystarczającym narzędziem diagnostycznym.



Rysunek 1. Transmisja danych pomiędzy dwoma urządzeniami kablem USB z zastosowaniem protokołu UART

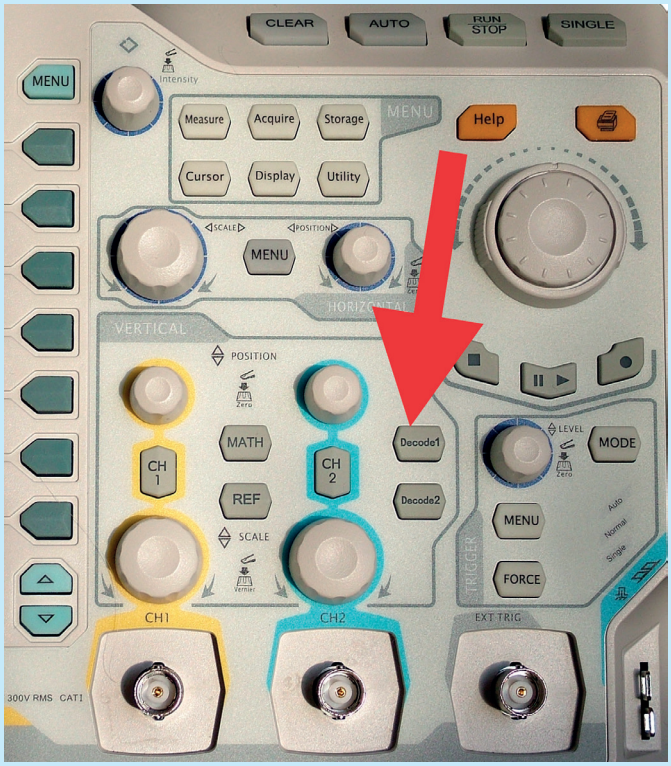
Teraz już na pewno możemy rozpocząć pomiary. Ramkę protokołu RS-232 przedstawiono na **rysunku 2**. Jak widać, mamy do czynienia z transmisją asynchroniczną, w której każdy wysyłany znak rozpoczyna się bitem startu. Następnie wysyłane są bity danych. Protokół przewiduje ramkę z 5, 6, 7, 8 lub 9 bitami. Najczęściej stosowana jest ramka 8-bitowa. W protokole przewidziano bit parzystości/nieparzystości, który jest wysyłany po bitach danych. Najczęściej jednak urządzenia z niego nie korzystają, w ramce po prostu go nie ma. Obowiązkowo natomiast muszą być wysłane bity stopu. Może być 1 bit, półtora bitu lub 2 bity. Zwykle korzystamy z jednego bitu stopu. Zaraz po zakończeniu wysyłania znaku (jego ostatniego bitu stopu) można rozpocząć wysyłanie następnego znaku. W nadajniku i w odbiorniku muszą być zastosowane wewnętrzne generatory taktujące o jednakowej częstotliwości. Ewentualna rozbieżność (w pewnym dopuszczalnym zakresie) jest niwelowana przez dosynchronizowanie się odbiornika na bitach startu. Warunkiem jest jednak prawidłowe rozpoznanie pierwszego bitu w transmisji, gdyż później bity startu niczym nie różnią się od pozostałych bitów i może nastąpić zerwanie komunikacji. Ważnym parametrem takiej komunikacji jest prędkość transmisji. Musi jej przestrzegać zarówno nadajnik, jak i odbiornik.



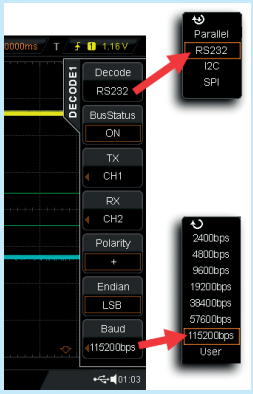
Rysunek 2. Ramka protokołu RS-232

Każde urządzenie dysponuje nadajnikiem (linia TxD) i odbiornikiem (linia RxD), co umożliwia transmisję dwukierunkową, więc obie linie muszą być skrzyżowane: TxD urządzenia 1 na RxD urządzenia 2 i analogicznie linie RxD na TxD.

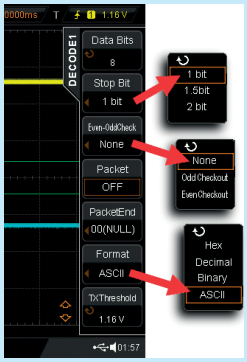
Powyższe informacje są wystarczające do odpowiedniego skonfigurowania obu urządzeń, będą też potrzebne do wprowadzenia prawidłowych nastaw analizatora protokołu. Przyjmijmy więc, że nasza ramka jest typu 115200,8,n,1, co oznacza, że dane są wysyłane z szybkością 115200 bitów na sekundę (to nie jest szybkość przesyłania znaków!), dana ma długość 8 bitów, nie korzystamy z bitu parzystości (gdyby taki bit był zastosowany zamiast „n” byłaby literka „o” – dla nieparzystości lub „e” – dla parzystości). Bit parzystości był wykorzystywany do kontroli prawidłowości transmisji, bardzo zawodnej zresztą. Specjalizowane układy odbiorników miały zain-



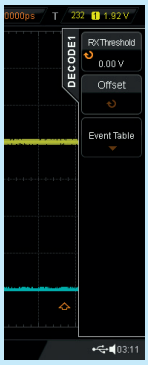
Fotografia 3. Lokalizacja przycisku *Decode1* na płycie czołowej oscyloskopu DS2202



Rysunek 4. Menu DECODE1 (pierwsza strona)



Rysunek 5. Menu DECODE1 (druga strona)



Rysunek 6. Menu DECODE1 (trzecia strona)

stalowany sprzętowy układ zliczający modulo 2 jedynki w przesyłanym znaku. Jeśli tak otrzymany wynik nie zgadzał się z bitem parzystości, ustawiana była odpowiednia flaga błędu, mogło być również generowane przerwanie. Dzisiaj opcja ta jest w zasadzie już tylko historyczną pozostałością i nikt z niej nie korzysta.

Nasz analizator protokołów będzie wykorzystywał oba kanały oscyloskopu, które dołączymy do linii TxD i RxD. *De facto* będziemy mierzyć interfejs UART z 3,3-woltową logiką. Pomiar rozpoczynamy od włączenia funkcji analizatora protokołów (przycisk *Decode1* (**fotografia 3**), – zauważmy, że możliwa jest niezależna analiza dwóch urządzeń). Na ekranie zostaje wyświetlone menu „DECODE1", w którym z łatwością odnajdujemy pola, odpowiadające parametrom transmisji (**rysunek 4**). Idąc od góry są to:

- typ protokołu –> wybieramy *RS232*,
- *BusStatus* – wybieramy *ON*, co powoduje włączenie dekodera odpowiedniego protokołu. Odczytane informację są wyświetlane na ekranie z zaznaczeniem linii, których dotyczą (np. TxD lub RxD),
- *TX* – wybieramy *CH1*, co oznacza, że kanał 1 oscyloskopu będzie dołączony do linii TxD,
- *RX* – wybieramy *CH2*, analogicznie dla linii *Polarity* – wybieramy +, gdyż badamy interfejs UART, pracujący w logice dodatniej. Gdyby to był RS-232, należałoby wybrać –,
- *Endian* – wybieramy *LSB*. W przypadku protokołu RS-232 w zasadzie nie mamy tu manewru, gdyż w ramce danych zawsze jako pierwsze są wysyłane bity najmniej znaczące,
- *Baud* – wybieramy *115200bps*, gdyż z taką właśnie szybkością pracuje badane urządzenie. Oprócz kilku wartości standardowych użytkownik może tu wprowadzać własną szybkość transmisji,

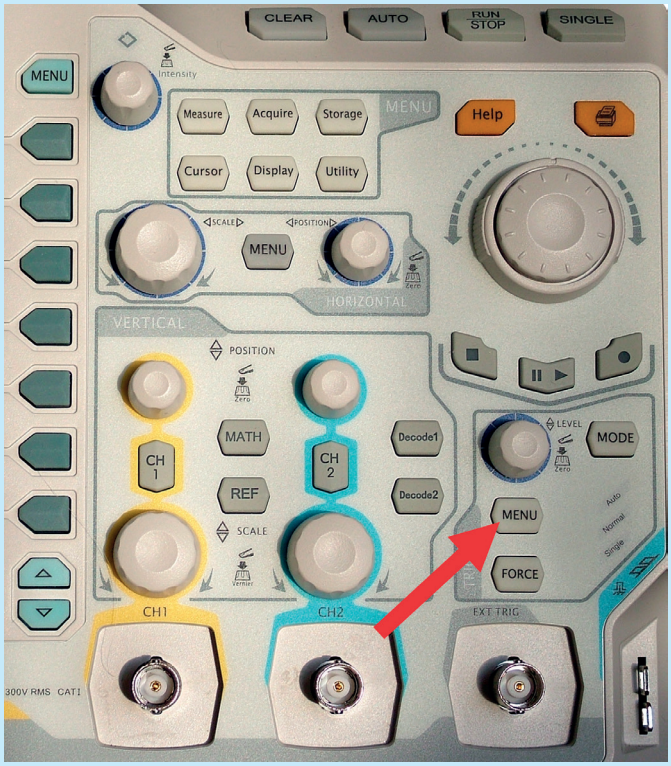
Teraz należy nacisnąć przycisk przejścia do następnej strony menu, czyli strzałkę w dół umieszczoną pod przyciskami funkcyjnymi obok ekranu. Rozwijają się kolejne opcje menu (**rysunek 5**).

- *Data Bits* – wybieramy *8*, gdyż pracujemy z ramką 8-bitową,
- *Stop Bit* – wybieramy *1*, bo w ramce przyjęliśmy jeden bit stopu,
- *Even-OddCheck* – wybieramy *None*, gdyż nie korzystamy z kontroli parzystości,
- *Packet* – na razie pomijamy tę opcję, wybieramy *OFF*,
- *PackeEND* – również pomijamy, nastawa nie ma znaczenia, jeśli wcześniej wybrano OFF. Opcje te będą omówione w dalszej części artykułu,
- *Format* – ponieważ w transmisji będą przesyłane informacje tekstowe wybieramy *ASCII*,
- *TXTherhold* – ten parametr określa poziom progowy dla zastosowanej logiki układów cyfrowych. Zwykle jest on ustawiany w połowie zakresu napięcia wyjściowego. Naciśnięcie przycisku umieszczonego przy opcji *TXTherhold* powoduje wyświetlenie poziomej linii na przebiegu związanym z linią TxD przedstawiającej aktualny poziom progowy. Wyświetlona jest także liczbowa wartość tego napięcia. Parametr *TXTherhold* jest zmieniany wielofunkcyjnym pokrętłem znajdującym się w górnym lewym rogu panelu zawierającego elementy regulacyjne.

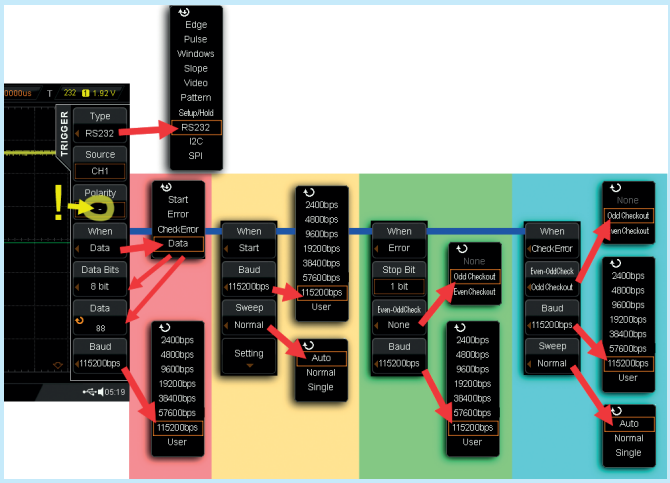
Przechodzimy do kolejnego menu, tak jak robiliśmy to wcześniej (**rysunek 6**).

- *RXTherhold* – wykonujemy analogiczne operacje dla linii RxD, tak jak robiliśmy to dla TxD,
- *Offset* – opcja ta pozwala dogodnie spozycjonować zdekodowane informacje. Korzystamy z pokrętła wielofunkcyjnego,
- *Event Table* – na razie pomijamy tę opcję.

Po wykonaniu wszystkich czynności, na ekranie powinien pojawić się oscylogram zawierający przebiegi na liniach TxD i RxD wraz ze zdekodowanymi danymi. Przebiegi będą jednak prawdopodobnie dość przypadkowo przesuwały się poziomo po ekranie, gdyż nie wprowadziliśmy jeszcze odpowiedniego warunku wyzwalającego. Procedura jest bardzo podobna jak podczas ustawianiu parametrów transmisji, z tym że teraz należy wybrać menu ukryte pod przyciskiem *Menu* w sekcji wyzwalania (**fotografia 7, rysunek 8**). Widzimy znajome opcje, którym powinny być nadane wartości zgodne z wprowadzonymi wcześniej parametrami. Jest jednak pewien niejasny wyjątek. Otóż w przypadku opcji *Polari-*



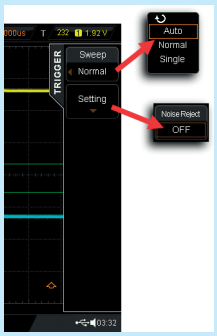
Fotografia 7. Lokalizacja przycisku *Menu* wybierającego opcje wyzwalania



Rysunek 8. Menu TRIGGER zawierające opcje wyzwalania (pierwsza strona)

ty definiowanej w menu warunku wyzwalania, trzeba wprowadzić wartość „–” (poprzednio był +). Prawdopodobnie jest to jakiś błąd w oprogramowaniu oscyloskopu DS2202. W omawianym menu pojawiła się dodatkowa pozycja:

- *When* – zawierająca opcje *Start*, *Error*, *CheckError* i *Data*. Ich znaczenie jest dość oczywiste. Po wybraniu opcji *Start*, układ wyzwalania współpracując z dekoderem transmisji będzie poszukiwał bitu startu i na nim nastąpi wyzwolenie. Problem polega na tym, że takich bitów zwykle jest dużo, rzadko kiedy wysyłamy jeden znak. Przy takiej nastawie trudno więc będzie uzyskać stabilny oscylogram. Oscyloskop będzie się synchronizował do każdego kolejnego napotykanego bitu startu. Opcja ta jest przydatna przy transmisji krótki paczek danych.



Rysunek 9. Menu TRIGGER (druga strona) z opcją *Sweep* ustalającą tryb pracy układu wyzwalania

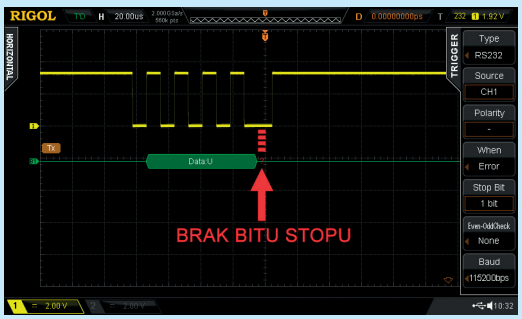
Przydatnym warunkiem wyzwalania jest natomiast ustawianie pułapki na określoną daną. W ten sposób łatwo można sprawdzać, czy dana taka pojawia się w ogóle w transmisji. Poszukiwana dana jest wprowadzania w postaci dziesiętnej w dolnym polu *Data* menu wyzwalania. Niestety, w oscyloskopie DS2202 przewidziano tylko taką postać danej, jeśli poszukujemy na przykład znaku ASCII „R”, to należy dokonać ręcznego przeliczenia, korzystając z tablicy kodów ASCII. W naszym przypadku znak „R” ma wartość dziesiętną 82, i taką liczbę należy wprowadzić, korzystając z pokrętła wielofunkcyjnego. Wybierając taki warunek wyzwalania, szczególnie wtedy, gdy poszukujemy specyficznego, rzadko występującego w transmisji zdarzenia, korzystne jest włączenie trybu wyzwalania „Normal”. Opcja ta jest dostępna po przejściu do kolejnego okna menu wyzwalania (**rysunek 9**). W trybie „Normal” wychwycone zdarzenie zostaje stabilnie zatrzaśnięte na ekranie. Wprawdzie w trybie „Auto” następowałoby także wyzwalanie na określonym zdarzeniu, ale przy dłuższym braku kolejnego takiego zdarzenia oscyloskop byłby wyzwalany bez spełnienia warunku, skutkujące zerwaniem synchronizacji.

Zastosowanie opcji *Error* powoduje wyzwalanie oscyloskopu po wykryciu błędu ramki, czyli braku bitu stopu. Sytuację taką przedstawiono na **rysunku 10.** Jednocześnie czerwony znacznik wyświetlony na ekranie sygnalizuje miejsce, w którym wystąpił błąd. Na oscylogramie widzimy, że tam gdzie powinien być znak stopu o wartości „1”, jest „0”.

Analogicznie działa opcja *CheckError*. Jest ona jednak stosowana tylko wtedy, gdy transmisja przebiega z włączoną kontrolą parzystości. Należy pamiętać o odpowiednim ustawieniu parametrów pracy analizatora protokołów.

Zakończyliśmy konfigurowanie naszego oscyloskopu do pracy jako analizatora protokołu RS-232. Możemy przystąpić do pomiarów, ale o tym już w następnym odcinku.

Jarosław Doliński, EP



Rysunek 10. Wyzwalanie po wykryciu błędu ramki